Agentic Workflows for Quant Development

Lessons learned from the QuantLibXlOil project

Kerstin Steinberg and Sebastian Schlenkrich[1],

FRAME Consulting GmbH, Berlin

In this document, we share our experiences in adopting agentic workflows for quant development projects. These experiences come from our work on QuantLibXlOil, a Python-based Excel add-in that exposes QuantLib functionality via xlOil. We believe that many observed aspects can be generalised. We encourage discussion and feedback from similar projects. The following three key lessons emerged from our work:

- Quality Over Quantity – Structure Enables Scalability
- Smart Delegation – Routine to AI, Responsibility to Humans
- Code Reviews Become More Critical – Not Obsolete

While AI tools like Copilot, Claude Code and Mistral promise productivity gains, their integration without clear structure and role allocation often introduces new challenges. The lessons learned above directly address these challenges and provide concrete recommendations for responsible AI use in development.

Project Context

The project aims to create a stable baseline version of a Python-based QuantLibXlOil Excel add-in using xlOil. It builds a uniform codebase as the foundation for regular reliable upgrade mechansim. This mechanism ensures consistent integration of future QuantLib updates while generating functional, standardized code.

We used Visual Studio Code with AI assistance from Copilot and Mistral, a European AI provider. Our development process involved analyzing and converting Python SWIG bindings into usable Python code and exposing them as Excel functions. Automated test suites were created and validated for both Python and Excel to ensure the technical reliability and accuracy of the results.

We plot a snapshot of a QuantLib/Excel sheet in Figure 1 to illustrate the context of the project.

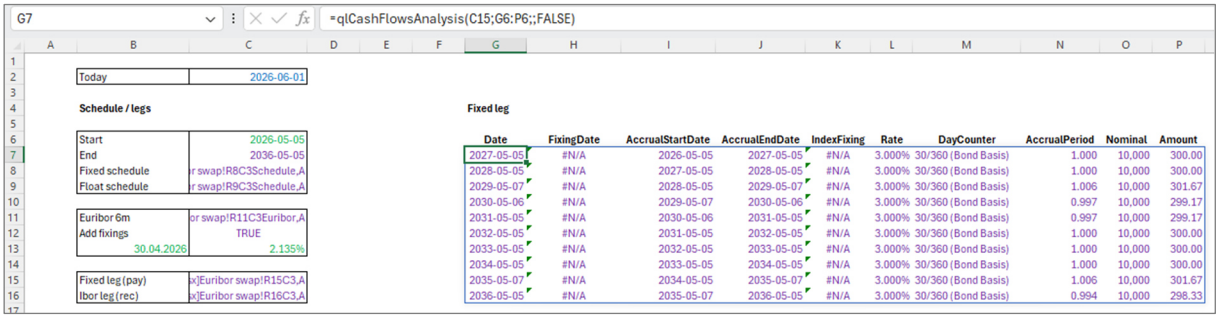


Figure 1: Snapshot of example QuantLib spread she

Lessons Learned

The table below presents the key experiences and lessons from the QuantLibXlOil-Excel add-in project side by side. The left column documents specific project observations and technical challenges. The right column summarizes the derived best practices and recommendations for sustainable software development.

[1] Contact: Sebastian Schlenkrich. Mobile: +49-1522-715-0780. Mail: sebastian.schlenkrich@frame-consult.de.

	Project Experience	Lessons Learned
Quality Over Quantity	The project has demonstrated one clear lesson from the start. A stable technical foundation must be established before scaling functionality. Validation and correction of Excel function bindings emerged as critical quality criteria from the project's earliest phases. A carefully planned phased rollout of QuantLib functions was significantly more efficient than attempting full library integration at once. It established a maintainable basis for future extensions. In reality, however, overly rapid progress at the project's start led to low-quality modules that the AI adopted as templates. The direct result was inconsistent Excel functions. Comprehensive reviews were indispensable, and cleanup of the binding code was required to ensure consistency across all exposed QuantLib functions.	Sustainable development starts with a solid foundation, not with writing as much code as possible. Clear structures, binding coding guidelines, precise requirements, and traceable documentation form this foundation. These elements reduce downstream errors, avoid unnecessary refactoring, and lower long-term development effort. Common rules should be defined before deploying AI. These include naming conventions, architectural guidelines and standardized testing. Reusable and proven skill files for tasks such as testing, or deployment further help standardize workflows and reduce repetitive coordination or prompts. Initial planning, documentation, and standardization pays off in the future. In a clear framework, AI operates reliably and reproducibly. Structure is not overhead but a prerequisite for speed and scalability.
Smart Delegation	The project showed that AI can reliably convert complex Python SWIG bindings into Excel functions. Thorough human validation remains essential to ensure correctness. A clear division of responsibilities has been proven crucial. AI handled repetitive tasks like binding conversion and basic Excel function generation. Developers managed architectural decisions. For example, they handled default values like QuantLib Calendar() constructor, unavailable in Python.	Productivity gains come from delegating the right tasks to AI. AI excels at recurring activities like standard implementations, test case creation, and applying existing patterns. Developers should focus on tasks requiring human judgment, such as architectural decisions and evaluating results. AI is a powerful tool within a human-led process for component creation from basic to complex. Precise requirements and quality criteria reduce subsequent correction effort.
Code Reviews	The project demonstrated AI's ability to generate Excel function bindings and unit test cases for the QuantLibXlOil-Excel add-in, making quality assurance critical. AI could create syntactically correct Excel functions. However, they sometimes contained functional errors or incorrect parameter types relative to the actual QuantLib implementation. In addition, the automatically generated unit test cases sometimes turned out to be incomplete. Effective reviews had to verify each function. Each function had to meet business requirements, align with the add-in's architectural guidelines, and integrate consistently with the existing function set.	Quality assurance becomes critical as AI adoption increases. AI-generated code may be syntactically correct and well-structured yet still contain functional errors, unhandled edge cases, or incorrect parameters. AI can misinterpret code comments, "TODO" markers, or ambiguous descriptions as implicit tasks, leading to unintended logic changes. Reviews must verify requirements, guidelines, architectural fit, and consistent implementation. Quality assurance combines automated tests, static analysis, and manual reviews. Human validation is essential to ensure that the code is functional, correct, secure, and maintainable.

Table 1: Comparison of Project Experiences and Derived Best Practices

Documentation and Source Code

Documentation and source code for the QuantLibXlOil package are hosted on GitHub at https://github.com/frame-consulting/QuantLibXlOil.